# B5000/B6000 SERIES SOFTWARE SYSTEM NOTES

## MARK 3.3

March 1982

Programming Activity
Mission Viejo Plant
Computer Systems Group
Burroughs Corporation
Mission Viejo, California

Proprietary Program Material

This material is proprietary to Burroughs Corporation and is not to be reproduced, used or disclosed except in accordance with program license or upon written authorization of the Patent Division of Burroughs Corporation, Detroit, Michigan 48232.

Copyright (C) 1982 Burroughs Corporation

Burroughs Corporation believes the information contained in this document to be accurate and reliable, and much care has been taken in its preparation. However, the corporation cannot accept responsibility, financial or otherwise, for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

# INTERPRETATIVE DMSII INTERFACE

Traditionally, programmatic access to DMSII has been through specially modified "host language" compilers. During compilation, DMSII statements are recognized and translated directly into calls on the database Accessroutines. Such an interface is static because all information needed to execute the statement at run-time must be known at compile-time. Static interfaces result in optimal run-time performance but have some disadvantages:

1. Because decisions about database operations must be made at compile-time, applications that need to postpone such decisions until run-time may be difficult to program.

2. Only languages that include DMSII syntax can access databases.

3. Interpretative languages such as LISP and APL have no concept of compile-time constructs and cannot access DMSII databases.
4. Programs become difficult to insulate from changes in database structure and often need re-compilation to avoid structural changes or to make use of new features.

This document describes an interpretative interface providing dynamic, run-time access to DMSII databases. At the expense of a small increase in processor overhead when connecting to DMSII, application programs may be written in an wider range of languages and become independent of the structure of particular databases.

The interpretative interface consists of entry points in a private library called the DMINTERPRETER. The library is tailored to a single database and generated by a new option in the BUILDINQ program. Using standard library entry point re-naming facilities, multiple databases and multiple invocations of one database may be accessed in a single program. DMINTERPRETER also provides functions not present in the host languages, including run-time determination of database structure, improved exception handling, and more sophisticated search algorithms.

## DMINTERPRETER Entry Points

DMINTERPRETER interfaces to ALGOL, COBOL, COBOL74, FORTRAN77, and PL/I are described here. Interfaces to other languages will be documented elsewhere as they become available.

Entry point names are formed by prefixing a host language verb with "DB". With a few exceptions, all library entry points take the same

number of parameters in the same order in all languages. The declared type of the parameters varies among languages but all contain the same information. Railroad diagrams describing the internal format of each parameter are presented in Appendix A. The syntax and semantics of the parameters to each entry point correspond as nearly as possible to the equivalent host language DMSII statement. Behavioral differences between DMINTERPRETER and the host language interface are described; similarities are not mentioned.

All entry points return an indication of exception through the procedure result unless otherwise noted. The declared type of the result varies among languages. Interpreting exception results is described in more detail below.

The DMINTERPRETER library is written in DMALGOL and tailored for a particular database during compilation. Entry points are provided that allow the library to be called from ALGOL, COBOL, COBOL74, FORTRAN77, and PL/I user programs. The DMINTERPRETER symbolic contains declarations of library entry points that can be incorporated into ALGOL, FORTRAN77, and PL/I user programs (COBOL and COBOL74 do not require explicit library declaration) via the INCLUDE compiler control option. The INCLUDE-able library declarations provide entry points that perform appropriate data type transfers of the parameters and procedure results from each language. Users may explicitly declare DMINTERPRETER entry points if they wish; this process is described in detail in a later section.

## Standard Entry Points

The following entry points correspond to functions available through the host language interface. The table accompanying each entry point shows the entry point identifer and the declared types of the returned result and parameters in each language. The result returned by each entry point is used for exception handling and is examined in more detail in a subsequent section. DISPLAY and COMP items should be declared at the 01 level in COBOL and COBOL74.
DISPLAY items should be alphanumeric; these items will be returned left-justified and blank-filled.
The programmer is responsible for converting these items to numeric when the data will be used for computational purposes.
Entry point names in FORTRAN77 may sometimes be shortened because of the 6-character identifier limitation.

DBOPEN ( <open disposition> )

<open disposition> indicates whether the database is to be opened UPDATE or INQUIRY. DBOPEN must be called before any database operations are performed. DBOPEN, the exception handling entry points, and operations on the intrinsic data set DBSTRUCTURE are the

only functions allowed when the database is closed.

| Language | Entry Point | Returned Result | <open disposition> |
|---|---|---|---|
| ALGOL | DBOPEN | BOOLEAN | STRING |
| COBOL | DBOPEN | Integer | DISPLAY |
| COBOL74 | DBOPEN | Integer | DISPLAY |
| FORTRAN77 | DBOPEN | LOGICAL | CHARACTER |
| PL/I | DBOPEN | BIT(48) | CHARACTER |

DBCLOSE

Closes the database. An implicit DBCLOSE is done at the end of the library execution if none has been previously done. DBCLOSE has no parameters.

| Language | Entry Point | Returned Result |
|---|---|---|
| ALGOL | DBCLOSE | BOOLEAN |
| COBOL | DBCLOSE | Integer |
| COBOL74 | DBCLOSE | Integer |
| FORTRAN77 | DBCLOS | LOGICAL |
| PL/I | DBCLOSE | BIT(48) |

DBBEGINTRANSACTION ( <audit> )

Enters transaction state. <audit> specifies whether an image of the current restart data set record will be audited. Calls on this entry point are ignored for unaudited databases.

| Language | Entry Point | Returned Result | <audit> |
|---|---|---|---|
| ALGOL | DBBEGINTRANSACTION | BOOLEAN | STRING |
| COBOL | DBBEGINTRANSACTION | Integer | DISPLAY |
| COBOL74 | DBBEGINTRANSACTION | Integer | DISPLAY |
| FORTRAN77 | DBBTR | LOGICAL | CHARACTER |
| PL/I | DBBEGINTRANSACTION | BIT(48) | CHARACTER |

DBENDTRANSACTION ( <auditsync> )

Leaves transaction state. <auditsync> determines whether the current restart data set record is audited and whether a sync point is forced. Calls on this entry point are ignored for unaudited databases.

| Language | Entry Point | Returned Result | <auditsync> |
|---|---|---|---|
| ALGOL | DBENDTRANSACTION | BOOLEAN | STRING |
| COBOL | DBENDTRANSACTION | Integer | DISPLAY |

| | | | |
|---|---|---|---|
| COBOL74 | DBENDTRANSACTION | Integer | DISPLAY |
| FORTRAN77 | DBETR | LOGICAL | CHARACTER |
| PL/I | DBENDTRANSACTION | BIT(48) | CHARACTER |

DBCREATE ( <data set> , <recordtype> )

Creates a record for the <data set>. <recordtype> specifies the variable format record type. An empty string or "0" creates a fixed length record or the fixed part of a variable format record.

| Language | Entry Point | Returned Result | <data set> | <recordtype> |
|---|---|---|---|---|
| ALGOL | DBCREATE | BOOLEAN | STRING | STRING |
| COBOL | DBCREATE | Integer | DISPLAY | DISPLAY |
| COBOL74 | DBCREATE | Integer | DISPLAY | DISPLAY |
| FORTRAN77 | DBCR | LOGICAL | CHARACTER | CHARACTER |
| PL/I | DBCREATE | BIT(48) | CHARACTER | CHARACTER |

DBRECREATE ( <data set> , <recordtype> )

Recreates a record for the <data set>. <recordtype> specifies the variable format record type. An empty string or "0" creates a fixed length record or the fixed part of a variable format record.

| Language | Entry Point | Returned Result | <data set> | <recordtype> |
|---|---|---|---|---|
| ALGOL | DBRECREATE | BOOLEAN | STRING | STRING |
| COBOL | DBRECREATE | Integer | DISPLAY | DISPLAY |
| COBOL74 | DBRECREATE | Integer | DISPLAY | DISPLAY |
| FORTRAN77 | DBRECR | LOGICAL | CHARACTER | CHARACTER |
| PL/I | DBRECREATE | BIT(48) | CHARACTER | CHARACTER |

DBSTORE ( <structure> )

Stores a record into the specified <structure>. Unlike the host languages, attempting a store outside transaction state on an audited database gives an exception. This behavior is an important improvement over the host language interface which terminates with an Invalid Operand fault.

| Language | Entry Point | Returned Result | <structure> |
|---|---|---|---|
| ALGOL | DBSTORE | BOOLEAN | STRING |
| COBOL | DBSTORE | Integer | DISPLAY |
| COBOL74 | DBSTORE | Integer | DISPLAY |
| FORTRAN77 | DBSTOR | LOGICAL | CHARACTER |
| PL/I | DBSTORE | BIT(48) | CHARACTER |

DBFREE ( <structure> )

Unlock a locked data set or global data record.

| Language | Entry Point | Returned Result | <structure> |
|---|---|---|---|
| ALGOL | DBFREE | BOOLEAN | STRING |
| COBOL | DBFREE | Integer | DISPLAY |
| COBOL74 | DBFREE | Integer | DISPLAY |
| FORTRAN77 | DBFREE | LOGICAL | CHARACTER |
| PL/I | DBFREE | BIT(48) | CHARACTER |

DBSET ( <structure> , <target> )

Performs the set to beginning, set to ending, and set to data set functions. When <structure> is a data set name, <target> indicates whether the data set should be set to beginning or ending. When <structure> is an index set name, <target> may indicate beginning or ending, or it may specify the data set that the index spans. In the latter case, the set is updated to point to the current data set record.

| Language | Entry Point | Returned Result | <structure> | <target> |
|---|---|---|---|---|
| ALGOL | DBSET | BOOLEAN | STRING | STRING |
| COBOL | DBSET | Integer | DISPLAY | DISPLAY |
| COBOL74 | DBSET | Integer | DISPLAY | DISPLAY |
| FORTRAN77 | DBSET | LOGICAL | CHARACTER | CHARACTER |
| PL/I | DBSET | BIT(48) | CHARACTER | CHARACTER |

DBFIND ( <direction> , <structure> , <condition> )

Performs the equivalent of a host language FIND statement. <direction> indicates the direction of the find - current, first, next, prior, or last. When non-empty, <condition> contains the selection expression used during the search. The complete syntax of <condition> is given in Appendix A. A DBFIND call is more sophisticated than a host language FIND statement. Several restrictions have been eased, and some algorithmic enhancements have been made. These changes are discussed in detail in a later section.

| Language | Entry Point | Returned Result | <direction> | <structure> | <condition> |
|---|---|---|---|---|---|
| ALGOL | DBFIND | BOOLEAN | STRING | STRING | STRING |
| COBOL | DBFIND | Integer | DISPLAY | DISPLAY | DISPLAY |
| COBOL74 | DBFIND | Integer | DISPLAY | DISPLAY | DISPLAY |
| FORTRAN77 | DBFIND | LOGICAL | CHARACTER | CHARACTER | CHARACTER |
| PL/I | DBFIND | BIT(48) | CHARACTER | CHARACTER | CHARACTER |

DBLOCK ( <direction> , <structure> , <condition> )

DBLOCK is exactly like DBFIND except that the data set record or

global data is left locked after being found.

| Language | Entry Point | Returned Result | <direction> | <structure> | <condition> |
|---|---|---|---|---|---|
| ALGOL | DBLOCK | BOOLEAN | STRING | STRING | STRING |
| COBOL | DBLOCK | Integer | DISPLAY | DISPLAY | DISPLAY |
| COBOL74 | DBLOCK | Integer | DISPLAY | DISPLAY | DISPLAY |
| FORTRAN77 | DBLOCK | LOGICAL | CHARACTER | CHARACTER | CHARACTER |
| PL/I | DBLOCK | BIT(48) | CHARACTER | CHARACTER | CHARACTER |

DBDELETE ( <direction> , <data set> , <condition> )

The parameters are identical to DBFIND, except that global data may not be deleted. DBLOCK is implicitly called to lock the data set record before it is deleted.

| Language | Entry Point | Returned Result | <direction> | <data set> | <condition> |
|---|---|---|---|---|---|
| ALGOL | DBDELETE | BOOLEAN | STRING | STRING | STRING |
| COBOL | DBDELETE | Integer | DISPLAY | DISPLAY | DISPLAY |
| COBOL74 | DBDELETE | Integer | DISPLAY | DISPLAY | DISPLAY |
| FORTRAN77 | DBDEL | LOGICAL | CHARACTER | CHARACTER | CHARACTER |
| PL/I | DBDELETE | BIT(48) | CHARACTER | CHARACTER | CHARACTER |

DBVERB ( <host statement> )

If the parameter evaluates to a complete host language statement, the requested operation is performed. The allowable forms of <host statement> are given in Appendix A. Most host language constructs are permitted but host language exception actions and input/output mapping clauses are not allowed.

DBVERB allows flexible, run-time definition of DMSII operations. But because it must completely parse its input, DBVERB runs at higher processor cost than the other, more explicit entry points.

| Language | Entry Point | Returned Result | <host statement> |
|---|---|---|---|
| ALGOL | DBVERB | BOOLEAN | STRING |
| COBOL | DBVERB | Integer | DISPLAY |
| COBOL74 | DBVERB | Integer | DISPLAY |
| FORTRAN77 | DBVERB | LOGICAL | CHARACTER |
| PL/I | DBVERB | BIT(48) | CHARACTER |

## Data Transfer Entry Points

The host language interface makes user work areas directly accessible to

COBOL, COBOL74 and PL/I programs. This is possible because the database description is known during compilation and used to enforce type checking and work area formats. However, DMINTERPRETER cannot allow the user to directly manipulate the work areas because type checking cannot be directly enforced and work area format may not be known.

Instead, DMINTERPRETER provides "get" and "put" entry points to move data to and from user work areas. This means that data will be correctly moved in and out of the work area but that the work area is not directly accessible to the program. "Get" entry points move data from the DMSII work area to user declared variables. "Put" entry points move data from user variables to DMSII work areas. Each item needed from a data set record must be individually moved to a user variable before it can be manipulated by the program.

The parameters of data transfer entry points are checked to ensure that the requested operation is compatible with the database description. For example, OCCURS items must be subscripted. The "put" entry points return an exception if <expr> does not fit into <item>, and the "get" entry points return an exception when <variable> is too short to contain the database <item>.

DBGETSTRING ( <structure> , <item> , <variable> )

Sets <variable> to a character string representing the contents of an ALPHA or GROUP item from the current <structure> record. <item> contains the name of the desired item from the work area.

| Language | Entry Point | Returned Result | <structure> | <item> | <variable> |
|---|---|---|---|---|---|
| ALGOL | DBGETSTRING | BOOLEAN | STRING | STRING | STRING |
| COBOL | DBGETDISPLAY | Integer | DISPLAY | STRING | STRING |
| COBOL74 | DBGETDISPLAY | Integer | DISPLAY | DISPLAY | DISPLAY |
| FORTRAN77 | DBGETC | LOGICAL | CHARACTER | CHARACTER | CHARACTER |
| PL/I | DBGETCHARACTER | BIT(48) | CHARACTER | CHARACTER | CHAR VAR |

DBGETREAL ( <structure> , <item> , <variable> )

Sets <variable> to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION <item> from the current <structure> record. If an exception occurs, <variable> is set to high-value (all bits on), and the appropriate exception is returned. For PL/I, FLOAT means BINARY FLOAT(38). NUMBER items that are too large to be retrieved by DBGETREAL can be retrieved using DBGETDOUBLE. COBOL and COBOL74 cannot access DBGETREAL but can retrieve the item value using DBDATA.

| Language | Entry Point | Returned Result | <structure> | <item> | <variable> |
|---|---|---|---|---|---|
| ALGOL | DBGETREAL | BOOLEAN | STRING | STRING | REAL |

| | | | | | |
|---|---|---|---|---|---|
| COBOL | not available | | | | |
| COBOL74 | not available | | | | |
| FORTRAN77 | DBGETR | LOGICAL | CHARACTER | CHARACTER | REAL |
| PL/I | DBGETFLOAT | BIT(48) | CHARACTER | CHARACTER | FLOAT |

DBGETDOUBLE ( <structure> , <item> , <variable> )

Sets <variable> to the numeric value of a NUMBER, REAL, FIELD, COUNT, RECORDTYPE, or POPULATION <item> from the current <structure> record. If an exception occurs, <variable> is set to high-value (all bits on), and the appropriate exception is returned. For PL/I, FLOAT means BINARY FLOAT(78). COBOL and COBOL74 cannot access DBGETDOUBLE but can retrieve the item value using DBDATA.

| Language | Entry Point | Returned Result | <structure> | <item> | <variable> |
|---|---|---|---|---|---|
| ALGOL | DBGETDOUBLE | BOOLEAN | STRING | STRING | DOUBLE |
| COBOL | not available | | | | |
| COBOL74 | not available | | | | |
| FORTRAN77 | DBGETD | LOGICAL | CHARACTER | CHARACTER | DOUBLE |
| PL/I | DBGETDOUBLE | BIT(48) | CHARACTER | CHARACTER | FLOAT |

DBGETBOOLEAN ( <structure> , <item> , <variable> )

Sets <variable> to the value of a BOOLEAN or BOOLEAN in a FIELD of bits in the current <structure> record. <item> contains the name of the desired item. <variable> is set to a false indication if an exception occurs. For COBOL and COBOL74, <variable> is set to the character string "TRUE" or "FALSE" depending upon the value of <item>. The character string returned may be truncated if <variable> is too short. PL/I programs use the SUBSTR function to isolate the right-most bit in the string ('1'B implies TRUE; '0'B means FALSE).

| Language | Entry Point | Returned Result | <structure> | <item> | <variable> |
|---|---|---|---|---|---|
| ALGOL | DBGETBOOLEAN | BOOLEAN | STRING | STRING | BOOLEAN |
| COBOL | DBGETBOOLEAN | Integer | DISPLAY | DISPLAY | DISPLAY |
| COBOL74 | DBGETBOOLEAN | Integer | DISPLAY | DISPLAY | DISPLAY |
| FORTRAN77 | DBGETL | LOGICAL | CHARACTER | CHARACTER | LOGICAL |
| PL/I | DBGETBIT | BIT(48) | CHARACTER | CHARACTER | BIT(48) |

DBPUTSTRING ( <structure> , <item> , <expr> )

Sets the value of <item> in the current <structure> record to the value obtained from <expr>. <item> must be declared in DASDL as ALPHA or GROUP. An exception is returned if <expr> does not produce a valid new value for <item>. <expr> must not exceed the length of the database <item> into which it is to be stored.

| Language | Entry Point | Returned Result | <structure> | <item> | <expr> |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| ALGOL | DBPUTSTRING | BOOLEAN | STRING | STRING | STRING |
| COBOL | DBPUTDISPLAY | COMP | DISPLAY | DISPLAY | DISPLAY |
| COBOL74 | DBPUTDISPLAY | COMP | DISPLAY | DISPLAY | DISPLAY |
| FORTRAN77 | DBPUTC | LOGICAL | CHARACTER | CHARACTER | CHARACTER |
| PL/I | DBPUTCHARACTER | BIT(48) | CHARACTER | CHARACTER | CHARACTER |

DBPUTREAL ( <structure> , <item> , <expr> )

Sets <item> in the current <structure> record to the new value indicated by <expr>. <item> must be a NUMBER, REAL, or FIELD item. If <expr> produces an incorrect new value, an exception is returned. For PL/I, FLOAT means BINARY FLOAT(38). The DBPUTDOUBLE entry point can be used to put items whose number of significant digits exceeds the capacity of a real variable. This entry point is not available to COBOL and COBOL74. Instead, the DBDATA interface can be used by COBOL and COBOL74 programs to put real values into these item types.

| Language | Entry Point | Returned Result | <structure> | <item> | <expr> |
|---|---|---|---|---|---|
| ALGOL | DBPUTREAL | BOOLEAN | STRING | STRING | REAL |
| COBOL | not available | | | | |
| COBOL74 | not available | | | | |
| FORTRAN77 | DBPUTR | LOGICAL | CHARACTER | CHARACTER | REAL |
| PL/I | DBPUTFLOAT | BIT(48) | CHARACTER | CHARACTER | FLOAT |

DBPUTDOUBLE ( <structure> , <item> , <expr> )

Sets <item> in the current <structure> record to the new value indicated by <expr>. <item> must be a NUMBER, REAL, or FIELD item. If <expr> produces an incorrect new value, an exception is returned. For PL/I, FLOAT means BINARY FLOAT(78). This entry point is not available to COBOL and COBOL74. Instead, the DBDATA interface can be used by COBOL and COBOL74 programs to put values into these item types.

| Language | Entry Point | Returned Result | <structure> | <item> | <expr> |
|---|---|---|---|---|---|
| ALGOL | DBPUTDOUBLE | BOOLEAN | STRING | STRING | DOUBLE |
| COBOL | not available | | | | |
| COBOL74 | not available | | | | |
| FORTRAN77 | DBPUTD | LOGICAL | CHARACTER | CHARACTER | DOUBLE |
| PL/I | DBPUTDOUBLE | BIT(48) | CHARACTER | CHARACTER | FLOAT |

DBPUTBOOLEAN ( <structure> , <item> , <expr> )

Sets the BOOLEAN <item> depending on evaluation of <expr>. <expr> must evaluate to FALSE or TRUE (character strings "FALSE" or "TRUE" for COBOL and COBOL74) to be valid. PL/I programs should place a '1'B or '0'B (representing TRUE and FALSE, respectively) in the right-most position of <expr>.

| Language | Entry Point | Returned Result | <structure> | <item> | <expr> |
|---|---|---|---|---|---|
| ALGOL | DBPUTBOOLEAN | BOOLEAN | STRING | STRING | BOOLEAN |
| COBOL | DBPUTBOOLEAN | Integer | DISPLAY | DISPLAY | DISPLAY |
| COBOL74 | DBPUTBOOLEAN | Integer | DISPLAY | DISPLAY | DISPLAY |
| FORTRAN77 | DBPUTL | LOGICAL | CHARACTER | CHARACTER | LOGICAL |
| PL/I | DBPUTBIT | BIT(48) | CHARACTER | CHARACTER | BIT(48) |

DBPUTNULL ( <structure> , <item> )

Sets <item> in the current <structure> record to its NULL value. <item> may not be of type BOOLEAN, COUNT, RECORDTYPE, or POPULATION.

| Language | Entry Point | Returned Result | <structure> | <item> |
|---|---|---|---|---|
| ALGOL | DBPUTNULL | BOOLEAN | STRING | STRING |
| COBOL | DBPUTNULL | Integer | DISPLAY | DISPLAY |
| COBOL74 | DBPUTNULL | Integer | DISPLAY | DISPLAY |
| FORTRAN77 | DBPUTN | LOGICAL | CHARACTER | CHARACTER |
| PL/I | DBPUTNULL | BIT(48) | CHARACTER | CHARACTER |

DBDATA ( <data request> , <data> )

<data request> contains a "get" or "put" operation as defined in Appendix A. <data> receives the value for "get" operations and is used to determine the new value for "put" operations. This entry point allows "get" and "put" commands to be dynamically constructed during program execution.

| Language | Entry Point | Returned Result | <data request> | <data> |
|---|---|---|---|---|
| ALGOL | DBDATA | BOOLEAN | STRING | STRING |
| COBOL | DBDATA | Integer | DISPLAY | DISPLAY |
| COBOL74 | DBDATA | Integer | DISPLAY | DISPLAY |
| FORTRAN77 | DBDATA | LOGICAL | CHARACTER | CHARACTER |
| PL/I | DBDATA | BIT(48) | CHARACTER | CHARACTER VARYING |

## Exception Handling Entry Points

All DMINTERPRETER entry points except DBEXCEPTIONTEXT and DBEXCEPTIONNAME return an indication of exception through a returned value. In ALGOL, FORTRAN77, and PL/I, DMINTERPRETER entry points return a 48-bit item interpreted as BOOLEAN in ALGOL, LOGICAL in FORTRAN77, and BIT(48) in PL/I. This item is the standard DMSII exception word. Fields in the item include the exception category and subcategory numbers and the number of the structure involved. The exception may be

captured when the entry point is called or may be accessed through the DBSTATUS function.

Exceptions are handled somewhat differently in COBOL and COBOL74. The entry points return an integer to the program. The compiler then maps this integer into whatever data type the user provided in the GIVING clause. Consequently, the user is free to declare the receiving item as desired. The returned value is one (1) if an exception occurred and zero (0) otherwise. The DBSTATUS entry point must be subsequently called to retrieve the actual exception. DBSTATUS returns the exception via its parameter.

| | |
|---|---|
| DBSTATUS | ALGOL, PL/I, FORTRAN77 |
| DBSTATUS ( <exception> ) | COBOL, COBOL74 |

Returns the exception result of the last DMINTERPRETER entry point called. For COBOL and COBOL74, <exception> must be declared as 01 COMP item with PIC 9(12). <exception> contains the same information as the boolean result word returned to other languages but in a format compatible with COBOL and COBOL74 data types. The format of <exception> used for COBOL and COBOL74 is shown in the COBOL and COBOL74 user program examples later in this document. User programs do not need to directly examine the contents of <exception>; they should just pass it on to DBEXCEPTIONTEXT.

| Language | Entry Point | Returned Result | <exception> |
|---|---|---|---|
| ALGOL | DBSTATUS | BOOLEAN | not used |
| COBOL | DBSTATUS | not used | COMP-2 |
| COBOL74 | DBSTATUS | not used | COMP |
| FORTRAN77 | DBSTAT | not used | LOGICAL |
| PL/I | DBSTATUS | BIT(48) | not used |

DBEXCEPTIONTEXT ( <exception> , <text> )

For an <exception> returned by DBSTATUS or directly by a DMINTERPRETER entry point, DBEXCEPTIONTEXT returns in <text> a full description of the exception. <text> contains "NO EXCEPTION" if no exception occurred. If <text> is not long enough to hold the entire message, the message text will be truncated as necessary to fit in <text>. Otherwise, <text> is padded with trailing blanks. The returned result for ALGOL and PL/I is always FALSE; the returned type is required for compatibility with PL/I.

| Language | Entry Point | Returned Result | <exception> | <text> |
|---|---|---|---|---|
| ALGOL | DBEXCEPTIONTEXT | BOOLEAN | BOOLEAN | STRING |
| COBOL | DBEXCEPTIONTEXT | Not used | COMP-2 | DISPLAY |
| COBOL74 | DBEXCEPTIONTEXT | Not used | COMP | DISPLAY |
| FORTRAN77 | DBEXCT | Not used | LOGICAL | CHARACTER |
| PL/I | DBEXCEPTIONTEXT | BIT(48) | BIT(48) | CHARACTER VARYING |

DBEXCEPTIONNAME ( <exception> , <text> )

For an <exception> returned by DBSTATUS or directly by a DMINTERPRETER entry point, DBEXCEPTIONNAME returns in <text> the name of the type of exception that occurred. <text> may contain the name of any DMSII exception category, such as NOTFOUND or OPENERROR. <text> contains "NO EXCEPTION" if no exception occurred. If <text> is not long enough to hold the entire message, the message text will be truncated as necessary to fit in <text>. Otherwise, <text> is padded with trailing blanks. The ALGOL and PL/I entry points always return FALSE; the procedure type is required for compatibility with PL/I.

| Language | Entry Point | Returned Result | <exception> | <text> |
|---|---|---|---|---|
| ALGOL | DBEXCEPTIONNAME | BOOLEAN | BOOLEAN | STRING |
| COBOL | DBEXCEPTIONNAME | Not user | COMP-2 | DISPLAY |
| COBOL74 | DBEXCEPTIONNAME | Not used | COMP | DISPLAY |
| FORTRAN77 | DBEXCN | Not used | LOGICAL | CHARACTER |
| PL/I | DBEXCEPTIONNAME | BIT(48) | BIT(48) | CHARACTER VARYING |

DMINTERPRETER entry points can return any exception returned by the host language interface. An additional category of exceptions has been added for errors peculiar to the interpretive interface. The new category and subcategory numbers and their corresponding texts are listed in Appendix B.

## DBFIND Enhancements

The DBFIND entry point performs the same functions as the host language FIND statement. However, several significant improvements have been made to the search algorithms in the DMINTERPRETER. Some search optimization techniques previously available only in INQUIRY are now available for use in programming languages.

### Non-key Items in Selection Expressions

The DMINTERPRETER allows non-key items in <condition>s. This is an improvement over the host language interface which allows only key and key data items in conditions. Relations about non-key items might require a linear search, depending on any other relations that might be in the same condition. If an item is not REQUIRED, it may be tested for NULL. For NULL items, the relations "<item> NEQ <value>" and "<item> = NULL" are TRUE and all other relations are FALSE.

## Selection Expression Evaluation

Since it is interpretive, the DMINTERPRETER must determine how to process selection expressions at run-time. To reduce this overhead, the most recent DBFIND statement is saved in an internal form. Consequently, a selection expression must be re-evaluated only when it changes. As a result, successive calls on DBFIND which are identical involve less overhead than they would otherwise.

## Finding the Data Set with Selection Expressions

The <structure> parameter to DBFIND may specify a <data set> even when the <condition> is non-empty. The host language interface does not allow selection expressions on data sets. If the <data set> has no spanning sets or accesses declared for it, a linear search of the data set is performed. When spanning sets or accesses are available and the selection <condition> can be satisfied using an indexed search, the DMINTERPRETER chooses the set or access that provides the most efficient search and uses it instead of the data set. Subsets are not considered. As long as the search <condition> remains unchanged, DMINTERPRETER will continue to use the selected set instead of the data set. By using a <data set> instead of an <index set> in DBFIND, application programs can be protected from changes to the database that add or delete spanning sets.

When the <condition> changes, DMINTERPRETER re-evaluates its choice of index set used to access the data set. If a different index set is chosen, DBFIND could produce unexpected results. For example, a FIND NEXT operation might actually return a prior record. DMINTERPRETER returns an exception when the situation is detected. The problem can be avoided by any of the following:

1. Not changing the <condition> in the middle of a search on a data set.

2. Specifying the <index set> in DBFIND instead of a <data set>.

3. Calling DBSET or DBFIND with <direction> equal to FIRST or LAST before changing the <condition>.

## Early Termination of Linear Searches

When the host language interface determines that a binary search cannot be used, it generates a linear search. The DMINTERPRETER must also do linear searches in some cases, but it does not always search the entire structure being used. In many cases,(e.g. a LEQ on an ASCENDING key), the search can be terminated the first time the <condition> fails. This form of linear search may often be faster than a binary search.

## More Robust Binary Searches

To binary search a complex key, the host language interface requires an equality test for all but the most minor key specified. Otherwise, a linear search of the index is performed. DMINTERPRETER can generate a binary search if all conditions which do not use EQL use the same binary searchable inequality.

For example, given a set with PARTNO and QUANTITY as its first and second keys,

The following host language statement produces a linear search.

```
FIND PART AT PARTNO > 10 AND QUANTITY > 20;
```

The equivalent DMINTERPRETER statement

```
DBFIND(EMPTY,"PART","PARTNO>10 AND QUANTITY>20");
```

is processed using a binary search.

DMINTERPRETER also performs binary searches on items inside a GROUP key. The following example demonstrates the situation:

```
DASDL:  GROUPDATA DATA SET
          (GROUPKEY GROUP(GROUPITEM NUMBER(2);
                          OTHERITEM ALPHA(10))));
        GROUPSET SET OF GROUPDATA KEY IS GROUPKEY;
```

The host language statement:

```
FIND NEXT GROUPSET AT GROUPITEM = 99;
```

produces a linear search of GROUPSET.

The equivalent DMINTERPRETER statement

```
DBFIND("NEXT","GROUPSET","GROUPITEM=99");
```

is processed using a binary search of GROUPSET.

## Mixed Mode Searches

DMINTERPRETER may combine linear and binary searches to satisfy a <condition>. A binary search might be used to find the beginning of a subsequent linear search. The linear search might then terminate early when the <condition> goes FALSE.

## Key Area Optimization

The actual movement of a data set record into the user work area is not done as a result of all DBFIND statements. Instead, if a set is used, only the read on the set entry is done and the data set read operation is simply marked as "pending." DMINTERPRETER will complete the read if a statement is issued which requires (1) data which is

not in the key area, or (2) the path of the data set to be correct. This means that a DBFIND statement that references only key or key data items behaves like a host language FIND KEY OF statement. "Get" operations that reference only key or key data items will not cause the data set record to be read. Consequently, the data set record might never need to be read.

## Determining the Database Structure

With the introduction of an interpretive interface to DMSII, the structure of the database may not be known when a program is written. In this case, it must be determined by the program at run-time. The DMINTERPRETER provides an intrinsic, read-only data set called DBSTRUCTURE to hold structural information. Unlike normal data sets, it may be accessed before the database is opened.

DBSTRUCTURE is generated when the DMINTERPRETER is compiled. The user does not need to declare it in the DASDL description of the database. No sets or subsets span DBSTRUCTURE, and the only valid operations are DBFIND, DBGET and DBSET.

The DBSTRUCTURE intrinsic data set is optional; it may be omitted by using the SELECTED DATASETS option in BUILDING without selecting DBSTRUCTURE. If the database contains an actual data set called DBSTRUCTURE, generation of the intrinsic data set is suppressed. However, the user declared DBSTRUCTURE data set cannot be accessed when the database is closed.

DBSTRUCTURE behaves as if it had the following DASDL description:

```
DBSTRUCTURE DATA SET

  (DB-ID                      FIELD(16)       REQUIRED;
   DB-OWNER                   FIELD(16)       REQUIRED;
   DB-NAME                    ALPHA(17)       REQUIRED;
   DB-TYPE                    ALPHA(17)       REQUIRED;

   % ITEMS PRESENT WHEN DB-TYPE = "DATASET"

   DATASET-SUBTYPE            ALPHA(17);

   % ITEMS PRESENT WHEN DB-TYPE = "SET"

   SET-SUBTYPE                ALPHA(17);
   SET-ACCESS-METHOD          ALPHA(17);
   SET-SPANS-DATASET          NUMBER(4);
   % ITEMS PRESENT WHEN DB-TYPE = "ITEM"
   ITEM-SUBTYPE               ALPHA(17);
```

```
    ITEM-SUBSCRIPTS          NUMBER(2);
    ITEM-OCCURS              NUMBER(4);
    ITEM-OWNER-GROUP         NUMBER(4);
    ITEM-REQUIRED            BOOLEAN;
    ITEM-SIZE                NUMBER(4);
    ITEM-SCALE-FACTOR        NUMBER(2);
    ITEM-SIGNED              BOOLEAN;
    ITEM-KEY-CLASS           ALPHA(17);
    ITEM-RECORD-TYPE         NUMBER(3);

    % ITEMS PRESENT WHEN DB-TYPE = "LINK"

    LINK-TO-DATASET          NUMBER(4);
    LINK-OCCURS              NUMBER(4);
   );
```

DBSTRUCTURE contains an entry for each data set, set, subset, access, item, and link in the database. The first four items in the record are prefixed by "DB" and have non-NULL values in every DBSTRUCTURE record. All other items in DBSTRUCTURE are named with a value for DB-TYPE as a prefix. They are have non-NULL values in a given record if the value of DB-TYPE in that record corresponds to their prefix. For example, if DB-TYPE is "LINK", only the items prefixed by "LINK" will have non-NULL values. Sets which have GROUP keys have two entries in the DBSTRUCTURE: one which lists the group as its key and another which lists the elementary items as its keys. Each database will also have an entry for global data. This entry can be used to find all of the disjoint structures and global data items in the database.

DB-ID

A positive number which uniquely identifies each component in the database. DB-ID is used to allow records in DBSTRUCTURE to be associated with each other. Items in more than one variable record type are considered to be different entities.

DB-OWNER

Contains the DB-ID of the data set which owns this entry. DB-OWNER is zero for the global data entry. All disjoint structures and global data items point to the global data entry.

DB-NAME

The name of the entity padded with trailing blanks.

DB-TYPE

Contains a string describing the type of the entity.

| | |
|---|---|
| DATASET | indicates a data set. |
| SET | indicates a set, subset, or access. |
| ITEM | indicates a data item. |

LINK indicates a link

DATASET-SUBTYPE

A string describing the type of data set. Valid values are STANDARD, RANDOM, ORDERED, UNORDERED, GLOBAL, DIRECT, RESTART, COMPACT, and STRUCTURE (for DBSTRUCTURE itself).

SET-SUBTYPE

Describes the type of set. Valid values are SET, SUBSET, and ACCESS.

SET-SEARCH-METHOD

Contains the accessing method used.

| | |
|---|---|
| DIRECT | for DIRECT data set accesses |
| BINARY | for ORDERED data set accesses and INDEX SEQUENTIAL and ORDERED LIST sets |
| HASH | for RANDDOM data set accesses and INDEX RANDOM sets |
| LINEAR | for UNORDERED data set accesses and BIT VECTOR sets |

SET-SPANS-DATASET

The DB-ID of the spanned data set.

ITEM-SUBTYPE

A string describing the type of item. Valid values are ALPHA, GROUP, FIELD, NUMBER, REAL, BOOLEAN, COUNT, TYPE, and POPULATION.

ITEM-SUBSCRIPTS

The number of subscripts required when using the item. If none are required, ITEM-SUBSCRIPTS is zero.

ITEM-OCCURS

If an item has an occurs clause, the subscript limit in its occurs clause is stored in this field. If ITEM-SUBSCRIPTS is zero, ITEM-OCCURS contains a NULL. To find the other subscript limits for multiply-subscripted items, the ITEM-OCCURS entries where DB-ID equals ITEM-OWNER-GROUP must be inspected. Items which have a non-zero value in ITEM-SUBSCRIPTS and a NULL in ITEM-OCCURS are non-occurring items within occurring groups.

ITEM-OWNER-GROUP

The DB-ID of the next higher group or field which contains this item. It can be used to determine the index limit for items in occurring groups and fields. It can also be used to determine which items belong to a given group or field. Items which do not belong to a group or field have a zero for ITEM-OWNER-GROUP.

ITEM-REQUIRED

TRUE if the entity is a REQUIRED item and FALSE otherwise.

ITEM-SIZE

Non-NULL for NUMBER, FIELD, ALPHA, and GROUP items only. Contains the declared length of the item in hexades, bits, bytes, and bytes, respectively. ITEM-SIZE does not include any space which may be necessary for the sign.

ITEM-SCALE-FACTOR

The declared number of digits to the right of the decimal point for NUMBERs and REALs that have a <scale factor> declared in DASDL. A NUMBER value may not exceed 10**((ITEM-SIZE)-(ITEM-SCALE-FACTOR)), and a REAL must be less than 549755813887.5/10**(ITEM-SCALE-FACTOR) unless the REAL was declared without a <precision>.

ITEM-SIGNED

TRUE if the entity is a NUMBER or REAL which may be negative.

ITEM-KEY-CLASS

A string containing the key status of the item in spanning sets.

| | |
|---|---|
| MAJOR | item is the major key of a spanning set or access |
| MINOR | item is not a major key of a spanning set or access, but it is a minor key |
| DATA | item only appears as key data in spanning sets or accesses |
| NONE | item does not appear in the key area of any access or spanning set |

ITEM-RECORD-TYPE

Contains the record type for the item. Zero for items in fixed records or the fixed part of variable format records. Since an item with a particular DB-ID can only be in one record type, any other item of the same name in a different record type has a unique DB-ID.

LINK-TO-DATASET

The DB-ID of the data set to which this link points.

LINK-OCCURS

The limit of the OCCURS clause for this link. A zero in LINK-OCCURS indicates that no OCCURS clause was present.

Pragmatics

---

The DMINTERPRETER has a constant, small processor time penalty as compared to the host language inteface. Observing the following rules when writing programs that use the interpretative interface can help keep DMINTERPRETER overheads low.

1. When programing in ALGOL or PL/I, use pre-initialized string variables instead of string expressions as parameters. This avoids the need to allocate string temporaries in the calling procedure.

2. DMINTERPRETER remembers the last structure on which it operated. Consequently, structure switching overheads can be avoided if consecutive operations are performed on the same structure.

3. Avoid passing unnecessary blanks to the DMINTERPRETER. They involve considerable extra overhead because additional procedures must be called and string temporaries must be allocated during blank elimination. For example, DBSTORE("DS") is much faster than DBSTORE(" DS").

4. Because ALGOL and PL/I programs directly call the entry points that perform the DMINTERPRETER functions, some overheads can be minimized by programming in these languages.

Since calls on DBBEGINTRANSACTION and DBENDTRANSACTION are ignored for unaudited databases, application programs can be simplified by unconditionally calling these entry points. The program can then be used against an audited or unaudited database without alteration.

To properly compile a PL/I program that invokes either DBGETDOUBLE or DBPUTDOUBLE, an OPTIONS(DOUBLE) clause should be used on the outer-most procedure declaration.

The DMINTERPRETER library stack uses globals to retain information between calls. This greatly increases the speed of execution, but prevents multi-processing. However, if the library declared by a parent task is visible to two or more offspring tasks, the offspring can get unexpected results if they call the library simultaneously. Locking conventions in the DMINTERPRETER could prevent such confusion but would slow down execution speed for the vast majority of users. Consequently, when the library is visible to multiple offspring tasks, users are responsible for implementing their own locking protocols.

As with other tailored DMSII software, database equation cannot be used when running the DMINTERPRETER. Instead, a separate DMINTERPRETER code file must be compiled for each database.

To see the SIB in a PROGRAMDUMP, the task options LIBRARIES and DBS must be set for the user program.

## Generating the DMINTERPRETER

The DMINTERPRETER is created using the BUILDINQ program. If BUILDINQ is run interactively, a new query is displayed:

```
WHICH SOFTWARE?

1 DMINQUIRY
2 DMINTERPRETER
```

If the response is 2, the DMINTERPRETER is generated. Refer to the DMSII Inquiry Reference Manual (form no. 5001472) for more information on running the BUILDINQ program.

The DMINTERPRETER symbolic is contained in the file called DATABASE/DMINTERPRETER. The generated DMINTERPRETER library code file is called DMINTERPRETER/<database>. The intermediate compiler input file is called DMIDIRECTORY/<database>.

The following WFL deck shows how the interpretative interface can be generated in a batch environment for a database called DMIDB.

```
?BEGIN JOB CREATE/DMINTERPRETER;
TASK T;
REMOVE DMIDIRECTORY/DMIDB;
RUN OBJECT/BUILDINQ CT];
DATA
$ NOZIP
DATABASE : DMIDB
DMINTERPRETER
? % --- END BUILDINQ INPUT ---
IF T ISNT COMPLETEDOK THEN
   ABORT "BUILDINQ RUN FAILED";
COMPILE DMINTERPRETER/DMIDB WITH DMALGOL CT] LIBRARY;
COMPILER FILE DASDL (TITLE=DMIDIRECTORY/DMIDB);
COMPILER FILE CARD  (KIND=PACK, PACKNAME=DISK,
                     TITLE=DATABASE/DMINTERPRETER);
IF T ISNT COMPILEDOK THEN
   ABORT "DMINTERPRETER COMPILE FAILED";
?END JOB.
```

## Entry Point Declarations

To ease the process of writing programs to use the DMINTERPRETER, DATABASE/DMINTERPRETER contains declarations of the library and its entry points for ALGOL, FORTRAN77, and PL/I. COBOL and COBOL74 does not

require declaration of libraries. The appropriate sequence range can be INCLUDEd for the language being used. The library identifier is DMI, and the entry point identifiers and parameter types are as shown above.

The INCLUDEd ranges declare all entry points. Users wishing to re-name DMINTERPRETER entry points or declare only a subset of the entry points must declare the entry points themselves. Since library and entry point declarations are separated in FORTRAN77 programs, the INCLUDE range contains only the entry points. FORTRAN77 programs must explicitly declare a library called DMI if the entry point declarations in the DMINTERPRETER symbolic are used. Since FORTRAN77 requires that the type of a function be declared in every block that references it, the DMINTERPRETER symbolic also contains type declarations for the FORTRAN77 entry points. The example FORTRAN77 program below shows the proper use of INCLUDEs in FORTRAN77.

| Language | Sequence Range |
|---|---|
| ALGOL | 20100000 - 20199999 |
| FORTRAN77 entry points | 20200000 - 20289999 |
| FORTRAN77 type declarations | 20290000 - 20299999 |
| PL/I | 20300000 - 20399999 |

The INCLUDE-able entry point declarations rename actual exported entry points so that the same function can be called by the same name in different languages. Because library entry points cannot be renamed in COBOL and COBOL74, the "DB" prefixed entry points are used for these languages. The renaming facilities in ALGOL, FORTRAN77, and PL/I are used to produce "DB" entry points in each of these languages. The ALGOL program example in the following section shows how entry points may be renamed. The majority of the actual work is performed by the ALGOL entry points. PL/I directly calls the ALGOL entry points. After translating their parameters to ALGOL-compatible constructs, the COBOL, COBOL74 and FORTRAN77 entry points call the ALGOL entry points to perform the actual request against the database.

Because the PL/I compile-time processor INCLUDE statement does not support inclusion of partial sequence ranges, the INCLUDE statement cannot be used. Instead, DMINTERPRETER entry points declarations can be inserted into PL/I programs from the DATABASE/DMINTERPRETER symbolic using the CANDE INSERT command.

The following table shows the names of all exported entry points and the corresponding ones available to each language. User's may use this table to construct their own DMINTERPRETER entry point declarations. For the data transfer entry points, some of the names presented with the above discussion of each entry point are not in this table. They are merely renamings of one of the following entry points. For example, the ALGOL entry point DBGETSTRING is a renaming of the ALGOLGETSTRING entry point.

| COBOL and COBOL74 | ALGOL and PL/I | FORTRAN77 |
|---|---|---|

Standard Entry Points

| | | |
|---|---|---|
| DBOPEN | ALGOLOPEN | FORTRANOPEN |
| DBCLOSE | ALGOLCLOSE | FORTRANCLOSE |
| DBBEGINTRANSACTION | ALGOLBEGINTRANSACTION | FORTRANBEGINTRANSACTION |
| DBENDTRANSACTION | ALGOLENDTRANSACTION | FORTRANENDTRANSACTION |
| DBCREATE | ALGOLCREATE | FORTRANCREATE |
| DBRECREATE | ALGOLRECREATE | FORTRANRECREATE |
| DBSTORE | ALGOLSTORE | FORTRANSTORE |
| DBFREE | ALGOLFREE | FORTRANFREE |
| DBSET | ALGOLSET | FORTRANSET |
| DBFIND | ALGOLFIND | FORTRANFIND |
| DBLOCK | ALGOLLOCK | FORTRANLOCK |
| DBDELETE | ALGOLDELETE | FORTRANDELETE |
| DBVERB | ALGOLVERB | FORTRANVERB |

Data Transfer Entry Points

| | | |
|---|---|---|
| DBGETDISPLAY | ALGOLGETSTRING | FORTRANGETCHARCTER |
| not available | ALGOLGETREAL | FORTRANGETREAL |
| not available | ALGOLGETDOUBLE | FORTRANGETDOUBLE |
| DBGETBOOLEAN | ALGOLGETBOOLEAN | FORTRANGETLOGICAL |
| DBPUTDISPLAY | ALGOLPUTSTRING | FORTRANPUTCHARACTER |
| not available | ALGOLPUTREAL | FORTRANPUTREAL |
| not available | ALGOLPUTDOUBLE | FORTRANPUTDOUBLE |
| DBPUTBOOLEAN | ALGOLPUTBOOLEAN | FORTRANPUTLOGICAL |
| DBPUTNULL | ALGOLPUTNULL | FORTRANPUTNULL |
| DBDATA | ALGOLDATA | FORTRANDATA |

Exception Handling Entry Points

| | | |
|---|---|---|
| DBSTATUS | ALGOLSTATUS | FORTRANSTATUS |
| DBEXCEPTIONTEXT | ALGOLEXCEPTIONTEXT | FORTRANEXCEPTIONTEXT |
| DBEXCEPTIONNAME | ALGOLEXCEPTIONNAME | FORTRANEXCEPTIONNAME |

## Example Programs

This ALGOL program is a simple browsing program to examine or modify the data in a database. The example program explicitly declares DMINTERPRETER entry points instead of INCLUDEing them from the symbolic.

```
BEGIN
   FILE RMT (KIND = REMOTE, MYUSE = IO, UNITS = CHARACTERS,
             MAXRECSIZE = 1920, MINRECSIZE = 1, FILETYPE = 3);
   BOOLEAN RSLT, RD;
   STRING VERB, DATA, MSG, NOTHING;
   LIBRARY DMI;
```

```
   BOOLEAN PROCEDURE DBEXCEPTIONTEXT(EXCEPTION,MSG);
      VALUE EXCEPTION;
      BOOLEAN EXCEPTION;
      STRING MSG;
      LIBRARY DMI (ACTUALNAME = "ALGOLEXCEPTIONTEXT");
   BOOLEAN PROCEDURE DBVERB(HOSTSTATEMENT);
      STRING HOSTSTATEMENT;
      LIBRARY DMI (ACTUALNAME = "ALGOLVERB");
   BOOLEAN PROCEDURE DBDATA(DATAREQUEST,DATASTRING);
      STRING DATAREQUEST, DATASTRING;
      LIBRARY DMI (ACTUALNAME = "ALGOLDATA");
   NOTHING := EMPTY;
   WRITE(RMT,<"#ENTER DATABASE NAME">);
   RD := READ(RMT,1920,VERB);
   IF NOT RD AND LENGTH(VERB) > 0 THEN
      DMI.TITLE := "DMINTERPRETER/" !! VERB !! ".";
   MSG := "#" !! VERB !! " READY";
   WRITE(RMT,LENGTH(MSG),MSG);
   RD := READ(RMT,1920,VERB);
   WHILE NOT RD DO
   BEGIN
      DATA := TAIL(HEAD(VERB,NOT " "),"  ");
      IF DATA = "GET" OR DATA = "PUT" THEN
      BEGIN
         IF DATA = "PUT" THEN
         BEGIN
            WRITE(RMT,<"#ENTER INPUT DATA">);
            RD := READ(RMT,1920,DATA);
            RSLT := DBDATA(VERB,DATA);
         END
         ELSE
         BEGIN
            RSLT := DBDATA(VERB,DATA);
            IF NOT RSLT THEN
            BEGIN
               MSG := "#DATA: " !! DATA;
               WRITE(RMT,LENGTH(MSG),MSG);
            END;
         END;
      END
      ELSE
         RSLT := DBVERB(VERB);
      IF NOT RSLT THEN
         WRITE(RMT,<"#">)
      ELSE
      BEGIN
         DBEXCEPTIONTEXT(RSLT,MSG);
         MSG := "#"!! MSG;
         WRITE(RMT,LENGTH(MSG),MSG);
      END;
      RD := READ(RMT,1920,VERB);
   END WHILE;
END.
```

The following COBOL and COBOL74 programs print the names of all employees in the PERSONNELDB database that speak Swedish as a native language. The first program is an example of COBOL using DMINTERPRETER; the second example is the same translated into COBOL74. Two items in the EMPLOYEE data set are relevant to the example: EMP-NAME, an ALPHA item 25 characters long containing the employee's name, and NATIVE-TONGUE, a 20 character ALPHA field containing the employee's native language. The DMINTERPRETER is used here because the search is performed on the non-key item NATIVE-TONGUE. The 01 item XCEPTION shows the format of the <exception> returned to COBOL and COBOL74 programs by DBSTATUS. Note the DISPLAY items that are passed as parameters must be declared at 01. Passing 77 level items will result in a parameter mismatch when linking to the DMINTERPRETER library.

Example of COBOL program

```
   IDENTIFICATION DIVISION.
   ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
   SELECT PRINT ASSIGN TO PRINTER.
   DATA DIVISION.
   FILE SECTION.
   FD PRINT.
   01 PRINT-BUF          PIC X(132).
   WORKING-STORAGE SECTION.
   01 FRST PIC X(5) VALUE "FIRST".
   01 NXT  PIC X(4) VALUE "NEXT".
   01 EMPLOYEE      PIC X(8)  VALUE "EMPLOYEE".
   01 CONDITION     PIC X(23)
                          VALUE "NATIVE-TONGUE='SWEDISH'".
   01 OPEN-TYPE     PIC X(7)  VALUE "INQUIRY".
   01 EMP-NAME      PIC X(8)  VALUE "EMP-NAME".
   01 DMI-NAME      PIC X(25)
                      VALUE  "DMINTERPRETER/PERSONNELDB".
   01 EMPLOYEE-NAME PIC X(25).
   01 RESULT        PIC 9(1) COMP-2.
   01 XCEPTION                COMP-2.
      05  STRUCTURE PIC 9(4) COMP-2.
      05  CATEGORY  PIC 9(3) COMP-2.
      05  SUB-CATEGORY PIC 9(4) COMP-2.
      05  FLAG         PIC 9(1) COMP-2.
   01 EXCEPTION-MESSAGE PIC X(200).
   01 EXCEPTION-NAME    PIC X(14).
   PROCEDURE DIVISION.
   DRIVER.
      CHANGE  ATTRIBUTE TITLE OF "DMINTERPRETER" TO DMI-NAME.
      CALL "DBOPEN OF DMINTERPRETER"
           USING OPEN-TYPE
           GIVING RESULT.
```

```
        IF RESULT = 1
           PERFORM DMEXCEPT.
        CALL "DBFIND OF DMINTERPRETER"
           USING FRST, EMPLOYEE, CONDITION
           GIVING RESULT.
        OPEN OUTPUT PRINT.
        PERFORM SPEAKER-SEARCH UNTIL RESULT = 1.
        PERFORM NOMORE.
     DONE.
        STOP RUN.
     DMEXCEPT.
        CALL "DBSTATUS OF DMINTERPRETER"
             USING XCEPTION.
        CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
             USING XCEPTION, EXCEPTION-MESSAGE.
        DISPLAY "UNEXPECTED DMSII EXCEPTION : " EXCEPTION-MESSAGE.
        CALL "DBCLOSE OF DMINTERPRETER"
             GIVING RESULT.
        PERFORM DONE.
     NOMORE.
        CALL "DBSTATUS OF DMINTERPRETER"
             USING XCEPTION.
        CALL "DBEXCEPTIONNAME OF DMINTERPRETER"
             USING XCEPTION,EXCEPTION-NAME.
        IF  EXCEPTION-NAME NOT EQUAL "NOTFOUND      "
            PERFORM DMEXCEPT.
        CALL "DBCLOSE OF DMINTERPRETER"
             GIVING RESULT.
        IF RESULT = 1
             PERFORM DMEXCEPT.
        PERFORM DONE.
     SPEAKER-SEARCH.
        CALL "DBGETDISPLAY OF DMINTERPRETER"
             USING EMPLOYEE,EMP-NAME,EMPLOYEE-NAME
             GIVING RESULT.
        IF RESULT = 1
             PERFORM DMEXCEPT.
        MOVE EMPLOYEE-NAME TO PRINT-BUF.
        WRITE PRINT-BUF AFTER ADVANCING 1.
        CALL "DBFIND OF DMINTERPRETER"
             USING NXT, EMPLOYEE, CONDITION
             GIVING RESULT.
```

Example of a COBOL74 program

```
     IDENTIFICATION DIVISION.
     ENVIRONMENT DIVISION.
     INPUT-OUTPUT SECTION.
     FILE-CONTROL.
     SELECT PRINT ASSIGN TO PRINTER.
     DATA DIVISION.
     FILE SECTION.
```

```
FD PRINT.
01 PRINT-BUF              PIC X(132).
WORKING-STORAGE SECTION.
01 FRST                   PIC X(5)  VALUE "FIRST".
01 NXT                    PIC X(4)  VALUE "NEXT".
01 EMPLOYEE               PIC X(8)  VALUE "EMPLOYEE".
01 CONDITION              PIC X(23)
                             VALUE "NATIVE-TONGUE='SWEDISH'".
01 OPEN-TYPE              PIC X(7)  VALUE "INQUIRY".
01 EMP-NAME               PIC X(8)  VALUE "EMP-NAME".
01 DMI-NAME               PIC X(25)
                             VALUE "DMINTERPRETER/PERSONNELDB".
01 EMPLOYEE-NAME          PIC X(25).
01 RESULT                 PIC 9(1)  COMP.
01 XCEPTION                         COMP.
   05 STRUCTURE           PIC 9(4)  COMP.
   05 CATEGORY            PIC 9(3)  COMP.
   05 SUB-CATEGORY        PIC 9(4)  COMP.
   05 FLAG                PIC 9(1)  COMP.
01 EXCEPTION-MESSAGE      PIC X(200).
01 EXCEPTION-NAME         PIC X(14).
PROCEDURE DIVISION.
DRIVER.
   CHANGE ATTRIBUTE TITLE OF "DMINTERPRETER" TO DMI-NAME.
   CALL "DBOPEN OF DMINTERPRETER"
      USING OPEN-TYPE
      GIVING RESULT.
   IF RESULT = 1
      PERFORM DMEXCEPT.
   CALL "DBFIND OF DMINTERPRETER"
      USING FRST, EMPLOYEE, CONDITION
      GIVING RESULT.
   OPEN OUTPUT PRINT.
   PERFORM SPEAKER-SEARCH UNTIL RESULT = 1.
   PERFORM NOMORE.
DONE.
   STOP RUN.
DMEXCEPT.
   CALL "DBSTATUS OF DMINTERPRETER"
      USING XCEPTION.
   CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
      USING XCEPTION, EXCEPTION-MESSAGE.
   DISPLAY "UNEXPECTED DMSII EXCEPTION: " EXCEPTION-MESSAGE.
   CALL "DBCLOSE OF DMINTERPRETER"
      GIVING RESULT.
   PERFORM DONE.
NOMORE.
   CALL "DBSTATUS OF DMINTERPRETER"
      USING XCEPTION.
   CALL "DBEXCEPTIONNAME OF DMINTERPRETER"
      USING XCEPTION, EXCEPTION-NAME.
   IF EXCEPTION-NAME NOT EQUAL "NOTFOUND      "
      PERFORM DMEXCEPT.
```

```
      CALL "DBCLOSE OF DMINTERPRETER"
         GIVING RESULT.
      IF RESULT = 1
         PERFORM DMEXCEPT.
      PERFORM DONE.
   SPEAKER-SEARCH.
      CALL "DBGETDISPLAY OF DMINTERPRETER"
         USING EMPLOYEE, EMP-NAME, EMPLOYEE-NAME
         GIVING RESULT.
      IF RESULT = 1
         PERFORM DMEXCEPT.
      MOVE EMPLOYEE-NAME TO PRINT-BUF.
      WRITE PRINT-BUF AFTER ADVANCING 1.
      CALL "DBFIND OF DMINTERPRETER"
         USING NXT, EMPLOYEE, CONDITION
         GIVING RESULT.
```

The following FORTRAN77 program plots two numeric items of any data set in any database against one another, given access to a plotting package.

```
    $ SET FREE
         BLOCK GLOBALS
   C*       THE DMI LIBRARY DECLARATION IS NOT IN THE FORTRAN77 INCLUDE
   C*       RANGE IN THE DMINTERPRETER SYMBOLIC, SO IT MUST BE DECLARED
   C*       BY THE USER PROGRAM
            LIBRARY DMI
         END
   C*    INCLUDE THE DMINTERPRETER ENTRY POINT DECLARATIONS
    $ INCLUDE "DATABASE/DMINTERPRETER" 20200000-20289999
   C*    INCLUDE THE PLOT PACKAGE SUBROUTINES
    $ INCLUDE "PLOTPACKAGE"
         SUBROUTINE EXCEPT(RESULT)
            LOGICAL RESULT
   C*    INCLUDE ENTRY POINT TYPE DECLARATIONS
    $ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999
            CHARACTER MSG *70
            CALL DBEXCT(RESULT,MSG)
            WRITE (6,10) MSG
     10     FORMAT(" UNEXPECTED DMSII EXCEPTION: ",A)
            STOP
         END
         SUBROUTINE NOMORE(RESULT,DONE)
            LOGICAL RESULT, DONE
            CHARACTER SNAME *18
            CHARACTER NF *18 /"NOTFOUND          "/
    $ INCLUDE "DATABASE/DMINTERPRETER" 20290000-202999999
            CALL DBEXCN(RESULT,SNAME)
            IF (SNAME .NE. NF) CALL EXCEPT(RESULT)
            IF (.NOT. DONE) GO TO 20
            WRITE(6,100)
     100    FORMAT(" NO VALUES TO PLOT")
            STOP
```

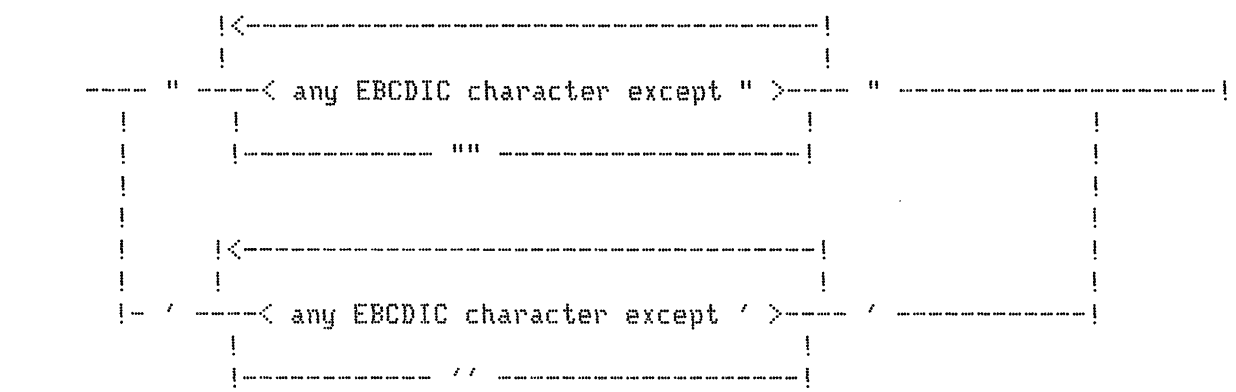
```
   20    CONTINUE
      END
C*    MAIN PROGRAM
         LOGICAL RESULT, BADGET
         INTEGER POINTS, SKIPS, I
         REAL    XVALUE, YVALUE
         CHARACTER
     *      FIRST   *5  /"FIRST"/,
     *      NEXT    *4  /"NEXT"/,
     *      BLANK   *1  /" "/,
     *      INQ     *7  /"INQUIRY     "/,
     *      DMINAM  *32,
     *      DBNAME  *17,
     *      STNAME  *17,
     *      XNAME   *17,
     *      YNAME   *17
 $ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999
C*       READ IN THE DATABASE, STRUCTURE, AND ITEM NAMES
         WRITE(6,100)
  100    FORMAT(" ENTER DATABASE NAME")
         READ (5,200) DBNAME
  200    FORMAT(A)
         WRITE(6,110)
  110    FORMAT(" ENTER STRUCTURE NAME")
         READ (5,200) STNAME
         WRITE(6,120)
  120    FORMAT(" ENTER X COORDINATE ITEM NAME")
         READ (5,200) XNAME
         WRITE(6,130)
  130    FORMAT(" ENTER Y COORDINATE ITEM NAME")
         READ (5,200) YNAME
C*       CONSTRUCT THE DMINTERPRETER CODE FILE TITLE
         DMINAM = "DMINTERPRETER/" // DBNAME // "."
         CHANGE(DMI,TITLE=DMINAM)
         RESULT = DBOPEN(INQ)
         IF (RESULT) CALL EXCEPT(RESULT)
         POINTS = 0
         SKIPS  = 0
C*       IPLOT INITIALIZES THE PLOT LIBRARY
         CALL IPLOT
         RESULT = DBFIND(FIRST,STNAME,BLANK)
         IF (RESULT) CALL NOMORE(RESULT,.TRUE.)
C*       A "DO 10 THRU 99 UNTIL NOTFOUND" LOOP FOLLOWS
  10     BADGET = DBGETR(STNAME,XNAME,XVALUE)
         IF (.NOT. BADGET) BADGET = DBGETR(STNAME,YNAME,YVALUE)
         IF (BADGET) GO TO 86
         CALL PLOT(XVALUE,YVALUE)
         POINTS = POINTS + 1
         GO TO 90
  86     SKIPS = SKIPS + 1
  90     RESULT = DBFIND(NEXT,STNAME,BLANK)
  99     IF (.NOT. RESULT) GO TO 10
         CALL NOMORE(RESULT,.FALSE.)
```

```
   C%        WPLOT WRITES THE PLOT
             CALL WPLOT
             WRITE(6,150) POINTS, SKIPS
    150      FORMAT(" ",I6," POINTS PLOTTED "/" ",I6," POINTS SKIPPED")
             STOP
           END
```

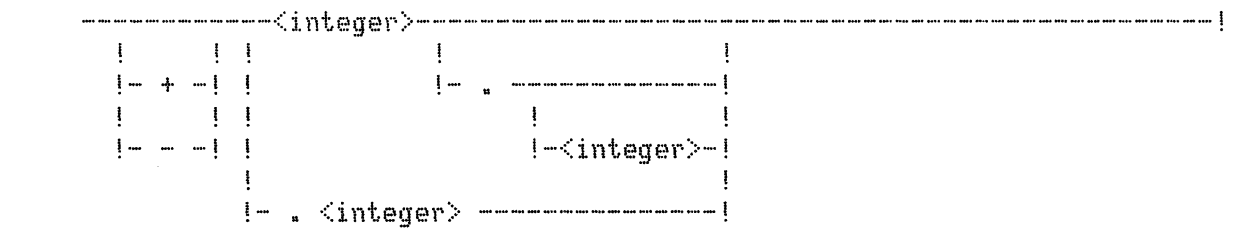
## Appendix A. Syntax Diagrams for Entry Point Parameters

<alpha item name>

<alpha literal>

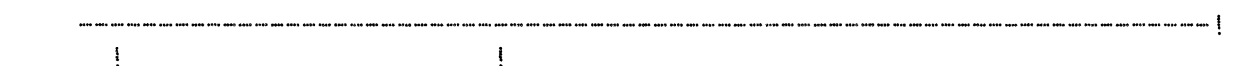


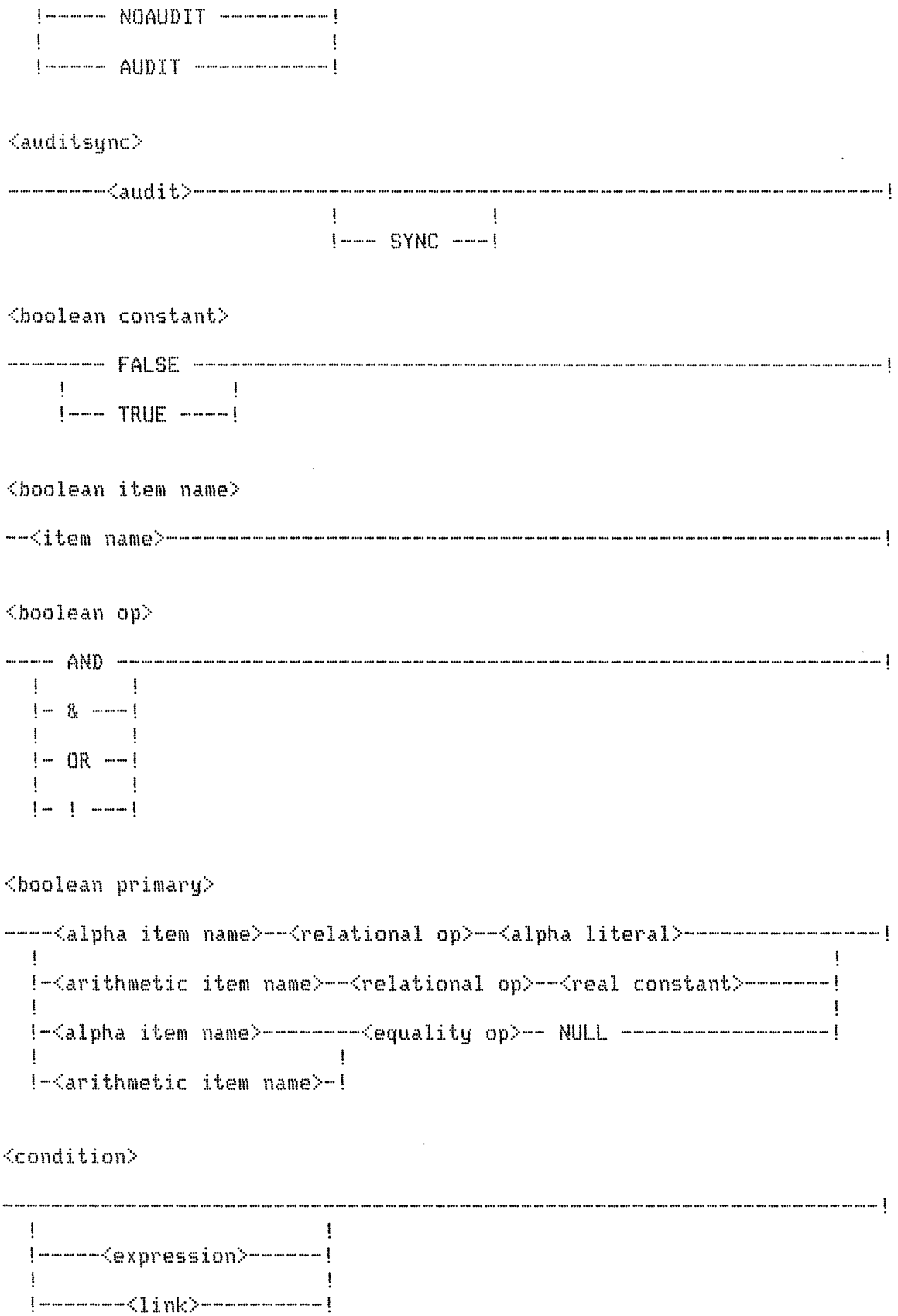

<arithmetic constant>



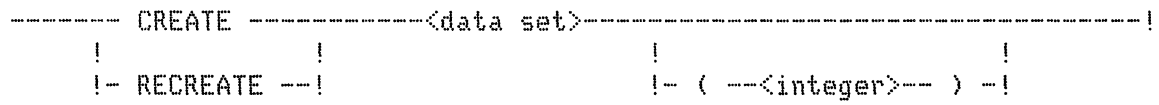

<arithmetic item name>

<audit>




NOAUDIT
AUDIT
<auditsync>
<audit>
SYNC
<boolean constant>
FALSE
TRUE
<boolean item name>
<item name>
<boolean op>
AND
&
OR
!
<boolean primary>
<alpha item name><relational op><alpha literal>
<arithmetic item name><relational op><real constant>
<alpha item name><equality op>
NULL
<arithmetic item name>
<condition>
<expression>
<link>

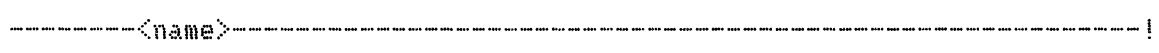
<create statement>




<data>

These forms of <data> apply only to "put" requests. For "get" requests, <data> should be a source language variable that will receive the value of the <item> named in the <data request>.

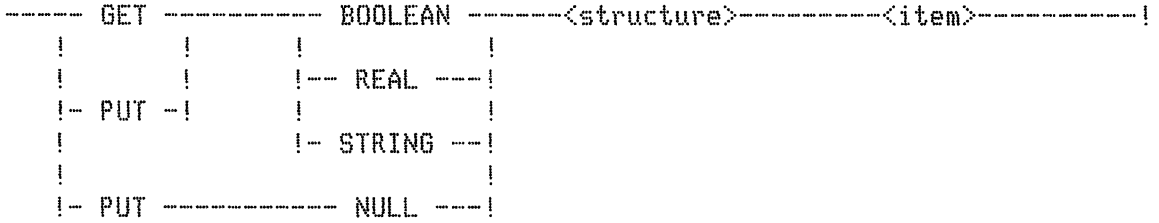
<database>



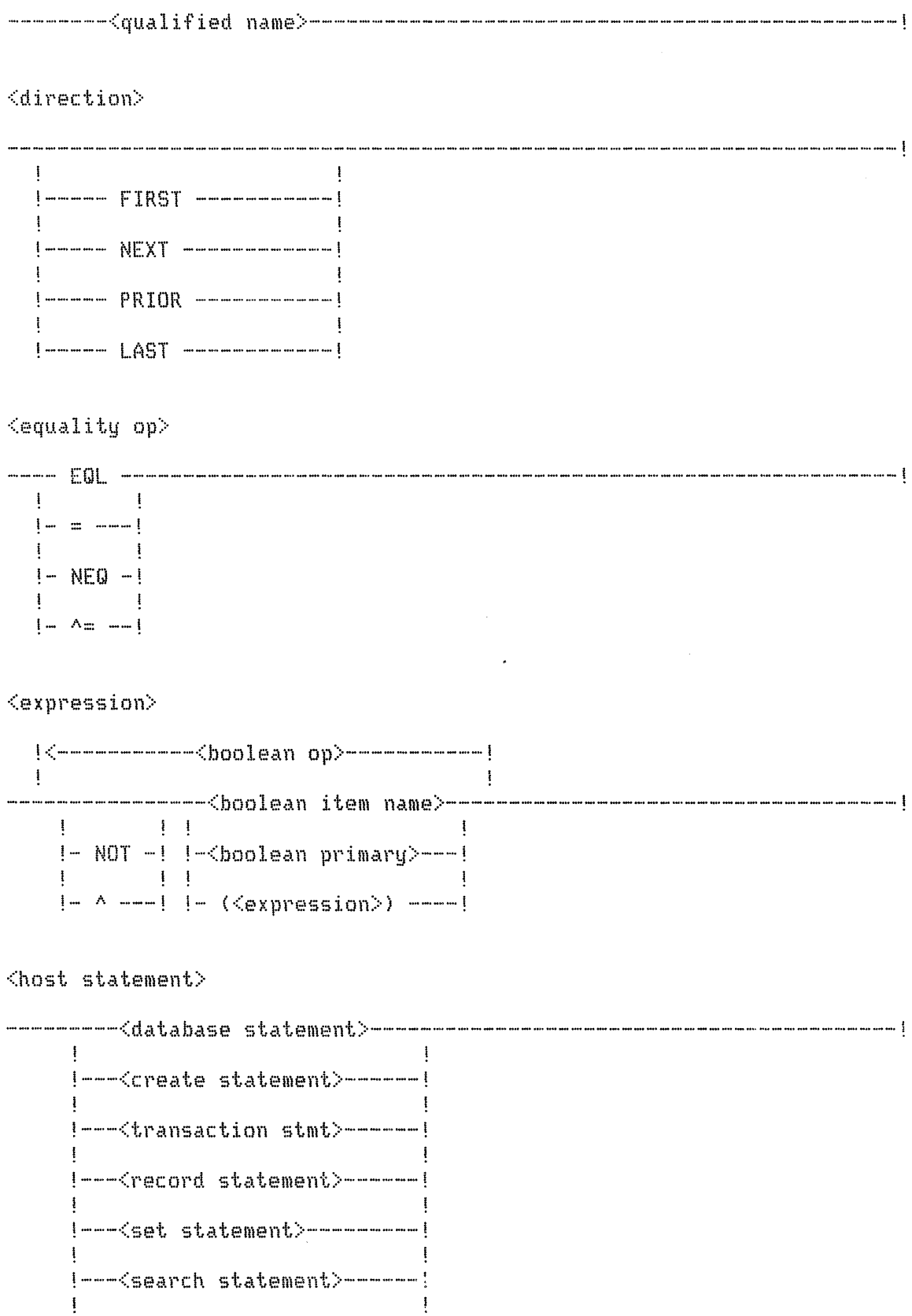

<database statement>

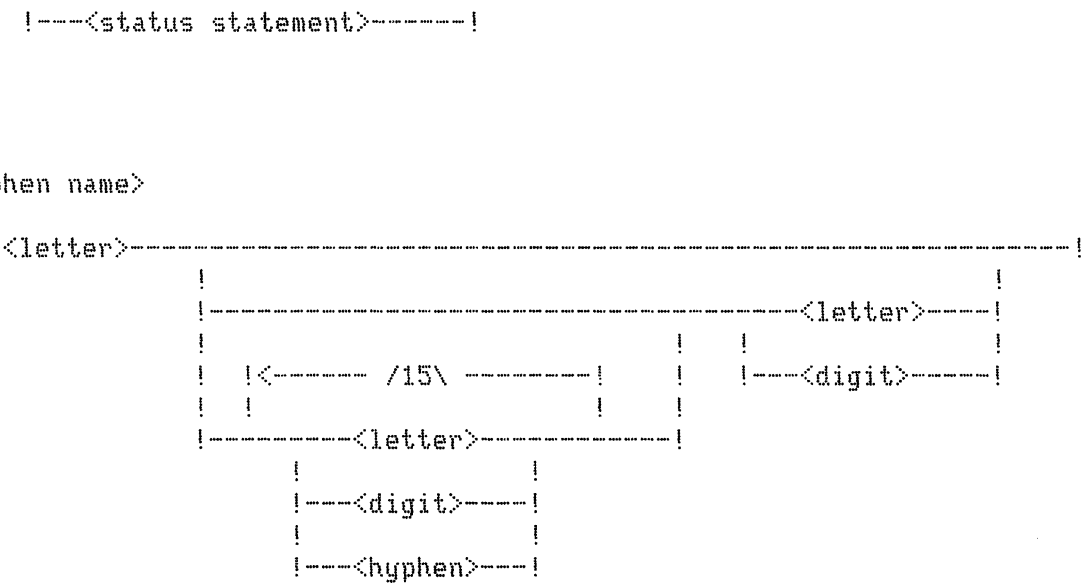
If the <database> is specified, it is checked for validity.

<data request>

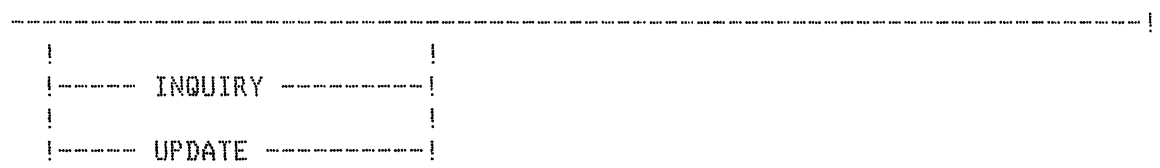



<data set>

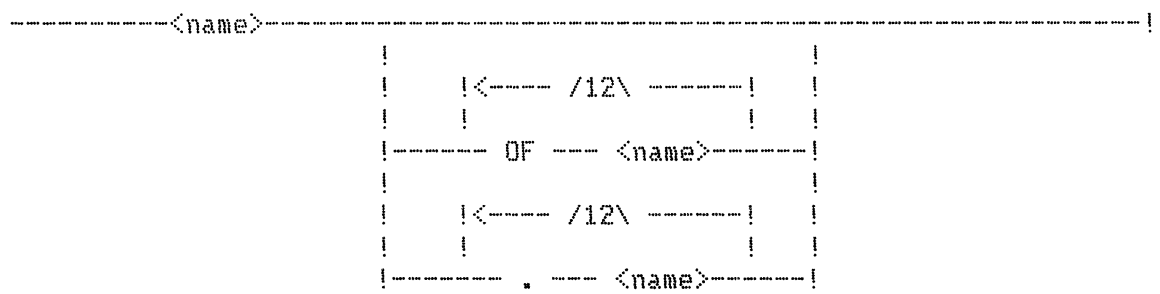

<qualified name>
<direction>
FIRST
NEXT
PRIOR
LAST
<equality op>
EQL
=
NEQ
^=
<expression>
<boolean op>
<boolean item name>
NOT
^
<boolean primary>
(<expression>)
<host statement>
<database statement>
<create statement>
<transaction stmt>
<record statement>
<set statement>
<search statement>


!---<status statement>-------!
<hyphen name>
-----<letter>
/15\
<letter>
<digit>
<letter>
<digit>
<hyphen>
<index set>
---------<qualified name>
<item name>
-----<name>
[ ---<integer>--- ]
( ---<integer>--- )
<link>
--------<qualified name>
[ --<integer>-- ]
( --<integer>-- )
<name>
----------<underscore name>
<hyphen name>

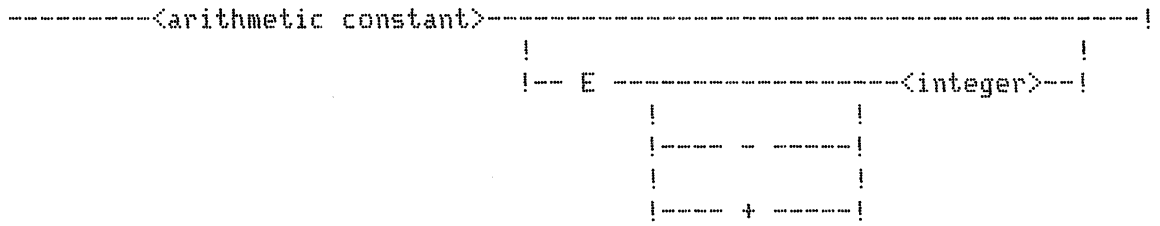
<open disposition>



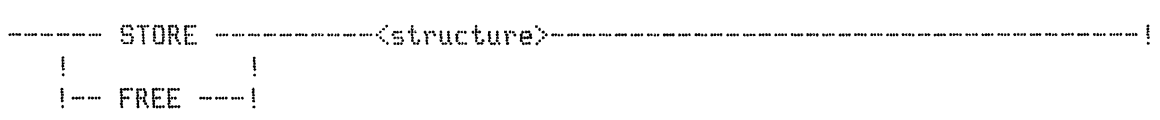

<qualified name>



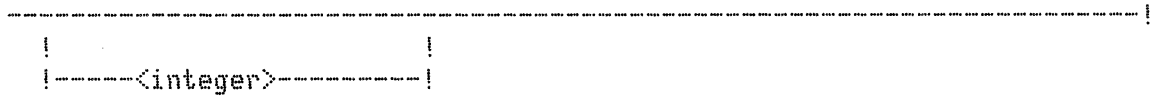

<real constant>



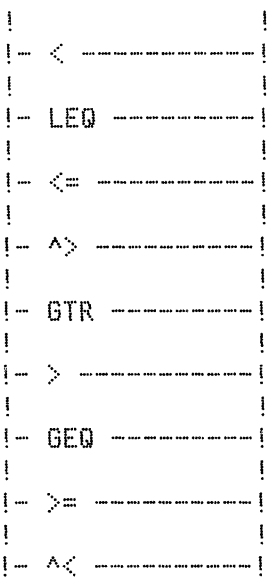

<record statement>



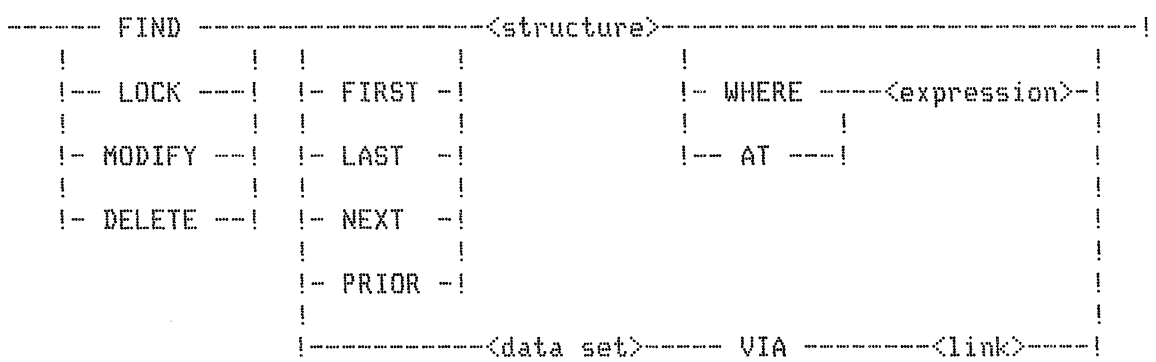

<recordtype>



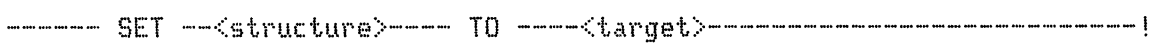

<relational op>




<search statement>



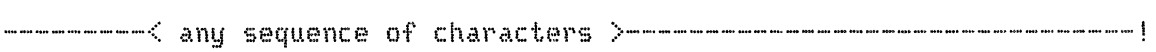

<set statement>




<status statement>

COBOL74 is not able to use the DMSTATUS form of a call on DBVERB because no <exception> parameter can be passed. Instead, these languages should directly call DBSTATUS if DBVERB indicates an exception occurred.

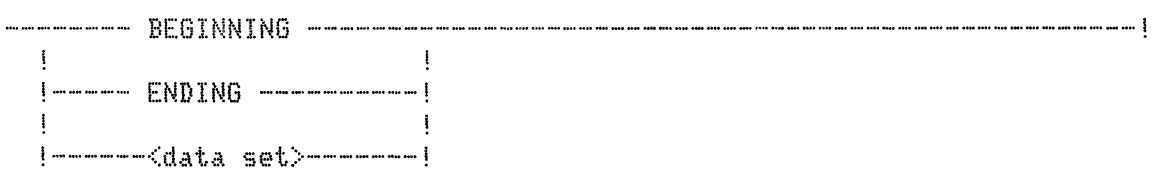
<string constant>



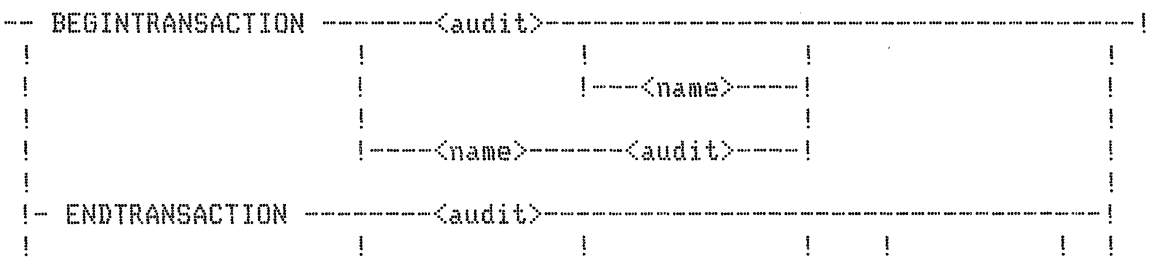
<structure> is used in the description of several entry points, but not all forms are valid for every entry point. The following table indicates what forms are valid for each entry point. The same rules apply when <structure> is used in a <host statement> or <data request>.

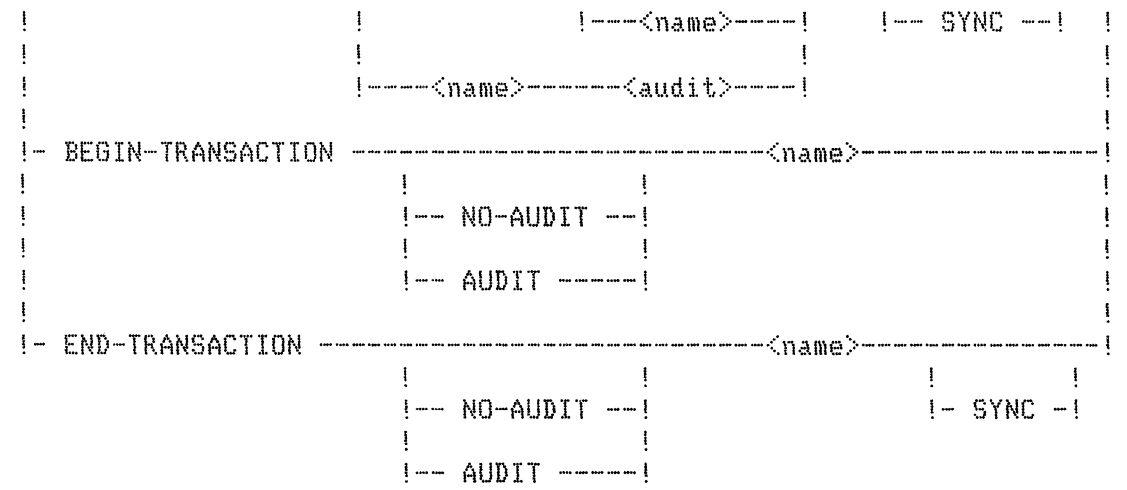
| Entry Point | <data set> | <index set> | <database> |
|---|---|---|---|
| DBSTORE | X | | X |
| DBFREE | X | | X |
| DBSET | X | X | |
| DBFIND | X | X | X |
| DBLOCK | X | X | X |
| DBGETSTRING | X | | X |
| DBGETREAL | X | | X |
| DBGETDOUBLE | X | | X |
| DBGETBOOLEAN | X | | X |
| DBPUTSTRING | X | | X |
| DBPUTREAL | X | | X |
| DBPUTDOUBLE | X | | X |
| DBPUTBOOLEAN | X | | X |
| DBPUTNULL | X | | X |

<target>



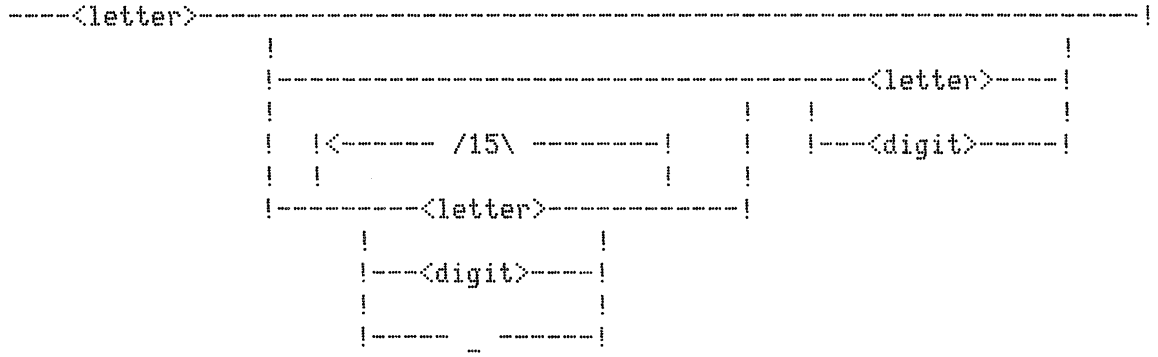

<transaction stmt>






<name> specifies the restart data set. When present, it is checked for validity.

<underscore name>




## Appendix B. Interpretative Interface Exceptions

New exceptions returned by DMINTERPRETER entry points are listed below. All exceptions fall into a new exception category called INTLIBERROR (category number 21). INTLIBERRORs are returned only by the interpretative interface.

| Subcategory | Text |
|---|---|

| | |
|---|---|
| 1 | First token of parameter to DBVERB/DBDATA not a valid verb |
| 2 | Specification string parameter not a valid key word |
| 3 | Structurename parameter is a DB id, but not the correct structure type |
| 4 | Structure or link name is not a known database id |
| 5 | Conflicting or insufficient qualification for structure or link name |
| 6 | Item name is not a database identifier |
| 7 | Item type does not correspond to the requested data type |
| 8 | Item name is not in the specified structure |
| 9 | Invalid index for an occurring item |
| 10 | Incorrect number of subscripts for an occuring item |
| 11 | The given item is not in the current variable record type or bad record type syntax |
| 12 | A negative value may only be put into a signed item |
| 13 | A get for this item found the value was null |
| 14 | The string parameter to a putstring must be the same length as the item, or may be shorter for alphas |
| 15 | The structure or link name was badly formed |
| 16 | The item name was badly formed |
| 17 | Bad literal appeared in a selection expression |
| 18 | Right parenthesis missing in selection expression |
| 19 | Relational operator expected in selection expression |
| 20 | A relation specified a null with the item required or the operator was not '=' or '^=' |
| 21 | The selection exp. did not end correctly or some unknown error occurred |
| 22 | A numeric argument to a put was invalid for the item |
| 23 | This operation is not valid for global data |

| | |
|---|---|
| 24 | Item and literal mis-matched length or type in a selection expression |
| 25 | Maximum number of relations in a selection expression is 47 |
| 26 | No valid spanning set for this embedded data set |
| 27 | Software failure in a selection expression tree execution |
| 28 | Invalid request while database is closed |
| 29 | Store or delete while not in transaction state |
| 30 | Index set left undefined by data set condition search |
| 31 | Parameter too short to hold response |
| 32 | Value of NUMBER item will not fit into real parameter |